Whare-Map: Heterogeneity in "Homogeneous" Warehouse-Scale Computers

Jason Mars
Univ. of California, San Diego
mars@cs.ucsd.edu

Lingjia Tang
Univ. of California, San Diego
lingjia@cs.ucsd.edu

Robert Hundt
Google
rhundt@google.com

ABSTRACT

Modern "warehouse scale computers" (WSCs) continue to be embraced as homogeneous computing platforms. However, due to frequent machine replacements and upgrades, modern WSCs are in fact composed of diverse commodity microarchitectures and machine configurations. Yet, current WSCs are architected with the assumption of homogeneity, leaving a potentially significant performance opportunity unexplored.

In this paper, we expose and quantify the performance impact of the "homogeneity assumption" for modern production WSCs using industry-strength large-scale web-service workloads. In addition, we argue for, and evaluate the benefits of, a *heterogeneity-aware* WSC using commercial web-service production workloads including Google's web-search. We also identify key factors impacting the available performance opportunity when exploiting heterogeneity and introduce a new metric, *opportunity factor*, to quantify an application's sensitivity to the heterogeneity in a given WSC. To exploit heterogeneity in "homogeneous" WSCs, we propose "*Whare-Map*," the **W**SC **H**eterogeneity Aw**are** **Map**per that leverages already in-place continuous profiling subsystems found in production environments. When employing "*Whare-Map*", we observe a cluster-wide performance improvement of 15% on average over heterogeneity–oblivious job placement and up to an 80% improvement for web-service applications that are particularly sensitive to heterogeneity.

1. INTRODUCTION

Warehouse-scale computers (WSCs) [7,18] are the class of datacenters that are designed, built, and optimized to run a number of large data-intensive web-service applications. Internet service companies such as Google, Microsoft, Amazon, Yahoo, and Apple spend hundreds of millions of dollars to construct and operate WSCs that provide web-services such as search, mail, maps, docs, and video [1,6,11,25]. This large cost stems from the machines themselves, power distribution and cooling, the power itself, networking equipment, and other infrastructure [14,15]. Improving the overall performance of jobs running in WSCs has been identified as one of the top priorities of web-service companies as it improves the overall cost efficiency of operating a WSC.

ISCA'13, Tel-Aviv, Israel.

Table 1: # of Machine Configs. in Google WSCs

D0	D1	D2	D3	D4	D5	D6	D7	D8	D9
4	3	2	3	2	3	2	5	2	2

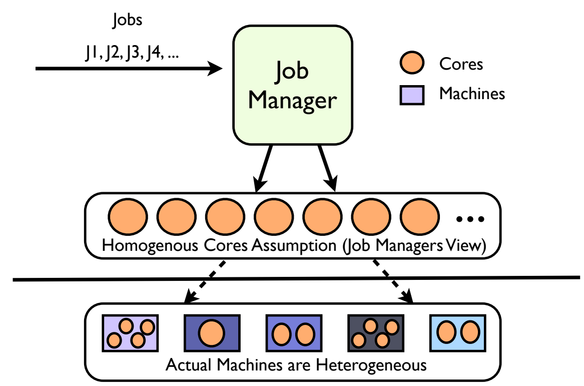

Figure 1: The Homogeneous Assumption - The Job Manager's View of a WSC

WSCs have been embraced as homogeneous computing environments [6–8]. However this is not the case in practice. These WSCs are typically composed of cheap and replaceable commodity components. As machines are replaced in these WSCs, new generations of hardware are deployed while older generations continue to operate. This leads to a WSC that is composed of a mix of machine platforms, i.e., a *heterogeneous* WSC. Table 1 shows the amount of platform diversity found in 10 randomly selected anonymized Google WSCs in operation. As shown in the table, these 10 datacenters house as few as two and as many as five different microarchitectural configurations, including both Intel and AMD servers from several consecutive generations. Yet, the assumption of homogeneity has been a core design philosophy behind the job management subsystems and system software stack of modern WSCs [7]. As Figure 1 shows, the job manager views a WSC as a collection of tens to hundreds of thousands of cores with the assumption of homogeneity. Available machine resources are assigned to jobs according to their core and memory requirements. The diversity across the underlying microarchitectures in a WSC is not explicitly considered by the job management subsystem. However, as we show in this work, ignoring this heterogeneity is costly, leading to inefficient execution of applications in WSCs.

While prior work [9,13,35,36] has acknowledged the hetero-

geneity in various types of datacenter systems, the homogeneous assumption is still widely adopted in modern WSCs due to 1) the limited understanding of the performance cost of this assumption in real commercial systems and 2) the lack of practical system design to exploit the heterogeneity in production.

There are two key insights to consider in understanding the heterogeneity present in emerging WSC system architectures:

1. The heterogeneity in WSCs differs than that found in a heterogeneous multicore chip, or the heterogeneity across processors in a single machine. In a WSC, it is the diversity in *execution environments* that must be considered. Broadly, we define an application's *execution environment* as the set of factors that can influence the execution of the application. In the scope of this paper, we focus this definition to the heterogeneity of the underlying processor microarchitecture coupled with the diverse possibilities of simultaneous co-running jobs on a given machine.

2. In the production WSC environments of large webservice companies such as Google, the system's hardware, software, and application stacks are co-designed for efficiency, and there is a set of key applications that run continually in these WSCs (such as websearch, maps, etc). This observation leads to an important insight. The performance opportunity present from the heterogeneity in machines are defined by the mix of applications that will run on these machines, and in turn, the performance opportunity present from the diversity in applications is defined by the particular mix of underlying machine configurations. As we vary either, the amount of performance opportunity changes significantly. As we show in this work, this can be formally quantified using a metric we call an application's *opportunity factor*.

These insights are prescriptive as to how we design a system to exploit this heterogeneity. Modern production WSC systems deploy continuous profiling runtimes such as the Google Wide Profiler (GWP) [27] that are run in production throughout the lifetime of the WSC. Currently these infrastructures are primarily used for retrospective analysis and offline tuning. However, as we argue in this work, these systems can and should be used to drive continuous online learning to steer heterogeneity analysis and adaptation in production. By harnessing these already in-place continuous monitoring capabilities, we demonstrate the efficacy of this approach with the design of the **W**SC **H**eterogeneity-aw**are Map**per, **Whare-Map**. Using Whare-Map, a novel extension to the core architecture of the WSC system design, we demonstrate how this heterogeneity can be exploited by leveraging in-place WSC monitoring subsystems.

Specifically, the contributions of this paper include:

- **Web-service Sensitivity to Heterogeneity:** We investigate the performance variability for large-scale web-service applications caused by the heterogeneity in "homogeneous" WSCs as it relates to *microarchitectural configurations* and *application co-locations* in production environments. We also introduce a novel metric, the *opportunity factor*, which quantifies how sensitive an application is to the heterogeneity. This metric characterizes the performance improvement potential for a given application when mapped intelligently in a given WSC.

- **Whare-Map:** We present *Whare-Map*, an extension to the current WSC architecture to exploit the emergent heterogeneity in WSCs. *Whare-Map* intelligently maps jobs to machines to improve the overall performance of a WSC. A required component of such approach is the ability to score and rank job-to-machine maps. We provide four map scoring policies that take advantage of the live monitoring services in modern WSCs, discuss the key tradeoffs between them, and perform a thorough evaluation in an experimental cluster environment.

- **Heterogeneity in Production:** We investigate the amount of heterogeneity present in a number of production WSCs running live Google Internet services including websearch, each housing thousands of nodes. We demonstrate the potential of *Whare-Map* by using it to quantify the performance opportunity from exploiting the heterogeneity in these production WSCs.

- **Factors Impacting Heterogeneity:** The rationale behind the homogeneity assumption stems from a lack of understanding of how the gradual introduction of diverse microarchitectural configurations and application types to a WSC impacts the performance variability. In this work, we also present a careful study of how varying the diversity in applications and machine types in a WSC affects how "homogenous" or "heterogeneous" a WSC becomes. We find that even a slight amount of diversity in these factors can present a significant performance opportunity. Based on our findings, we then discuss the tradeoffs for server purchase decisions and show that embracing heterogeneity can indeed improve the cost-efficiency of the WSC infrastructure.

A prototype of Whare-Map is evaluated on both a Google testbed composed of 9 large-scale production web-service applications and 3 types of production machines, as well as an experimental testbed composed of benchmark applications to provide repeatable experimentation.

Results Summary: This paper shows that there is a significant performance opportunity when taking advantage of emergent heterogeneity in modern WSCs. At the scale of modern cloud infrastructures such as those used by companies like Google, Apple, and Microsoft, gaining just 1% of performance improvement for a single application translates to millions of dollars saved. In this work, we show that large-scale web-service applications that are sensitive to emergent heterogeneity improve by more than 80% when employing *Whare-Map* over heterogeneity-oblivious mapping. When evaluating Whare-Map using our testbed composed of key Google applications running on three types of production machines commonly found co-existing in the same WSC, we improve the overall performance of an entire WSC by 18%. We also find a similar improvement of 15% in our benchmark testbed and in our analysis of production data from WSCs hosting live services.

Next, in Section 2 we discuss the background of our work. We then present a study of the heterogeneity in the WSC in Section 3. Section 4 presents our *Whare-Map* approach for exploiting heterogeneity in the WSC. We present an in-depth evaluation of the performance benefit of *Whare-Map* including a study in Google's production WSCs in Section 5. We present related work in Section 6, and finally, we conclude in Section 7.

2. BACKGROUND

In this section, we describe the job placement and online monitoring components that are core to the system architecture of modern WSCs.

workload	description
bigtable	A distributed storage system for managing petabytes of structured data
ads-servlet	Ads server responsible for selecting and placing targeted ads on syndication partners sites
maps-detect-face	Face detection for streetview automatic face blurring
search-render	Websearch frontend server, collect results from many backends and assembles html for user.
search-scoring	Websearch scoring and retrieval
protobuf	Protocol Buffer, a mechanism for describing extensible communication protocols and on-disk structures. One of the most commonly-used programming abstractions at Google.
docs-analyzer	Unsupervised Bayesian clustering tool to take keywords or text documents and "explain" them with meaningful clusters.
saw-countw	Sawzall scripting language interpreter benchmark
youtube-x264yt	x264yt video encoding.

Table 2: Production WSC Applications

2.1 Job Placement in WSCs

A WSC provides a view of the underlying system as a single machine with hundreds of thousands of cores and petabytes of memory. A *job* in a WSC is an application process that is typically long running, responsible for a particular sub task of a major service, and can generally be run on any machine within the WSC. Example jobs in a WSC include a result scorer for websearch, an image stitcher for maps, a compression service for video, etc. Job placement in the WSC is managed by a central job manager [7, 21, 30]. The job manager is a cluster-wide runtime that is tasked with mapping the job mix to the set of machine resources in a WSC, and operates independently of the OS. Each job has an associated configuration file that specifies the number of cores and memory required to execute the job. Based on the resource requirement, the job manager uses a bin-packing algorithm to place the job to a machine with the required resources available [21], after which a machine level manager (in the form of a daemon running in user-mode) uses *resource containers* [5] to allocate and manage the resources belonging to the task. The currently deployed job manager is unaware of machine heterogeneity and the potential benefits of intelligent job placement. We integrate our *Whare-Map* technique described in Section 4 with the job manager to conduct heterogeneity-aware mapping.

2.2 Live Monitoring in WSCs

For our WSC design to effectively exploit heterogeneity in job placement decisions, the job manager requires continuous feedback from prior placement decisions. At a minimum, a sampling of a service's performance on an assortment of platforms and co-runners is necessary. Fortunately, this rudimentary continuous cluster-wide monitoring capability is already deployed and available in state-of-the-art WSC platforms. For example, the Google Wide Profiler (GWP) [27] continuously profiles jobs and machines as they run in production and is deployed as a standard service in Google's entire production fleet. GWP provides a database and associated API that can be leveraged by software systems, such as the job manager, to query information about application placement, performance, and co-running jobs on a machine. As described later, our heterogeneity-aware technique uses the monitoring information provided by services such as GWP to conduct intelligent job-to-machine mappings.

3. HETEROGENEITY IN MODERN WSCS

The potential benefit of heterogeneity-awareness in WSCs hinges on the performance variability of applications across diverse microarchitectural configurations and co-runners. In this section, we investigate such performance variability for large-scale Internet service production applications. We focus on the performance sensitivity of applications to emergent heterogeneity and, equally importantly, the variance of this sensitivity itself across applications.

CPU	GHz	Cores	L2/L3	Name
Clovertown Xeon E5345	2.33ghz	6	8mb	**Clover**
Istanbul Opteron 8431	2.4ghz	6	6mb	**Istan**
Westmere Xeon X5660	2.8ghz	6	12mb	**West**

Table 3: Production Microarchitecture Mix

CPU	GHz	Cores	L2/L3	Memory
Core i7 920	2.67ghz	4	8mb	4gb
Core 2 Q8300	2.5ghz	4	4mb	3gb
Phenom X4 910	2.6ghz	4	6mb	4gb

Table 4: Experimental Microarchitecture Mix

In addition to the production applications, we also present results using an testbed composed of benchmark applications. Finally, we introduce a metric, *opportunity factor*, that, given the application mix and machine mix in a WSC, quantifies an application's sensitivity to heterogeneity within a closed eco-system of machine configurations and diverse applications.

3.1 Characterization Methodology

[**Google Testbed**] We first conducted our experiments using commercial applications (shown in Table 2) across three production platform types commonly found in Google's WSCs (shown in Table 3). The applications shown in the table cover nine large industry-strength workloads that are responsible for a significant portion of the cycles consumed in arguably the largest web-service WSC infrastructure in the world. Table 2 also presents a description for each application. Each application corresponds to an actual binary that is run in the WSC. These applications are part of a test infrastructure developed internally at Google composed of a host of Google workloads and machine clusters that have been both laboriously configured by a team of engineers for performance analysis and optimization testing across Google. Each application shown in the table operates on a repeatable log of thousands of queries of user activity from production. We use this test infrastructure throughout the remainder of this work. The number of cores used by each application is configured to three for both solo and co-location runs.

[**Benchmark Testbed**] To investigate how our findings using Google's infrastructure generalize to other application sets and to provide experimental results that are repeatable, we replicate our study in an experimental benchmark testbed. In our experimental infrastructure we use a spec-

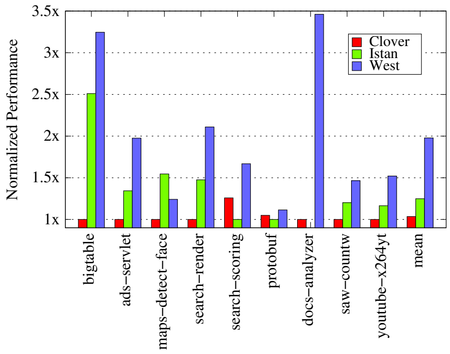

Figure 2: Performance comparison of key Google applications across three microarchitectures. (higher is better)

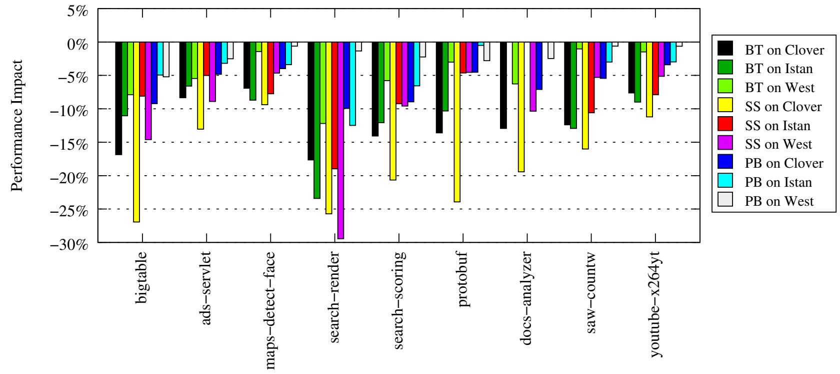

Figure 3: Google application performance when co-located with `bigtable` (BT), `search-scoring` (SS), and `protobuf` (PB). (negative indicates slowdown)

trum of 22 SPEC CPU2006 benchmarks on their `ref` input as our application types and three state of art microarchitectures as our machine types running Linux 2.6.29. The underlying microarchitectures of these three machine types are presented in Table 4. All application types are compiled with GCC 4.5 with O3 optimization.

3.2 Investigating Heterogeneity

[**Microarchitectural Heterogeneity**] We first characterize the performance variability due to microarchitectural heterogeneity in WSCs. In addition to quantifying the magnitude of the performance variability, our study also aims to investigate firstly, whether one microarchitectural configuration consistently outperforms others for all applications; and secondly, the variance of sensitivities across applications. As we discuss later in this section, the variance in performance sensitivity across a given application mix is indicative of the performance potential of adopting a heterogeneity-aware WSC design.

Figure 2 presents the experimental results for our Google testbed with 9 key Google applications (Table 2) running on 3 types of production machines (Table 3). The y-axis shows the performance (average instructions per second) of each application on three types of machines, normalized by the worst performance among the three for each application. `Docs-analyzer`'s data on Istanbul is missing because it is not configured for that particular platform.

Figure 2 shows that even among three architectures that are from competing generations, there is a significant performance variability for Google applications. More interestingly, no platform is consistently better than the others in this experiment. Although the Westmere Xeon outperforms the other platforms for most applications, `maps-detect-face` running on the Istanbul Opteron outperforms the Westmere Xeon by around 25%. On the other hand, the Clovertown Xeon and Istanbul Opteron compete much more closely. It is also important to note that even though the Westmere Xeon platform is almost always better than the other two, the performance sensitivity to platform types vary significantly across applications, ranging from gaining only 10% speedup for `protobuf` when switching from the worst platform (Opteron) to the best (Westmere Xeon), to as large as 3.5x speedup for `docs-analyzer`. This heterogeneity in performance sensitivity (various speedup ratios) impacts how job placement decisions should be made to maximize the overall performance and the amount of potential performance improvement achievable by intelligent mapping. To maximize the overall performance for a WSC composed of a limited number of each microarchitecture, a smart job manager should prioritize mapping those applications with higher speedup ratio to the faster machines. For example, to achieve the best overall performance, `docs-analyzer` or `big-table` should be prioritized to use the Westmere Xeon over `protobuf`. In Section 5.4 we delve into more details as what cause the performance variability by varying the workload mix.

[**Co-Runner Heterogeneity**] Figure 3 illustrates the performance variability due to co-location interference for Google applications. This figure shows the performance interference of each of the 9 Google applications when co-located with `bigtable` (BT), `search-scoring` (SS) and `protobuf` (PB). The y-axis shows the performance degradation of each benchmark when co-located on each platform. This degradation is calculated using the application's execution rate when co-located normalized to the execution rate when it is running alone on that platform. The lower the bar, the more severe the performance penalty. We observe that the same co-runner causes varying performance penalties to different applications with performance degradations ranging from close to no penalty, 2% or less in some cases, to almost 30%.

More interestingly, the heterogeneity in co-location penalty is not an isolated factor and is complicated by the heterogeneity in microarchitecture. As shown in the figure, for each application, the same co-running application may cause varying performance penalties on different microarchitectures. Also, microarchitectural heterogeneity on average has a more significant performance impact than co-location heterogeneity. While there is generally less than 30% performance degradation due to co-location, the performance variability due to microarchitectural heterogeneity is up to 3.5x. However, the relative impact of the two depends on applications. For some applications (e.g., `protobuf`), co-running heterogeneity has a greater impact than machine heterogeneity. The above observations imply that when exploiting the heterogeneity in WSCs to improve performance via better job-to-machine mapping, there may be a compounding benefit to consider both machine and co-runner heterogeneity simultaneously.

In addition to Google production applications, we also conducted similar experiments on our benchmark testbed. The results are presented in Figures 4 and 5. In summary, we observe that the amount of variability present from microarchitectural and co-runner diversity is significant for

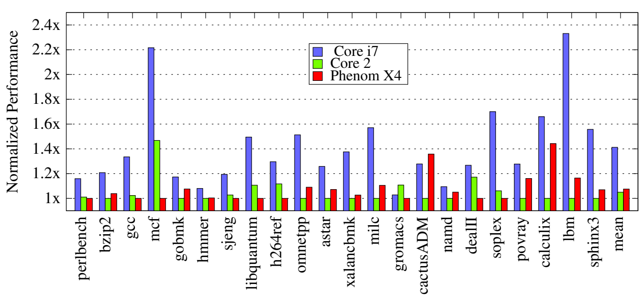

Figure 4: Performance comparison of benchmark workloads across three microarchitectures.

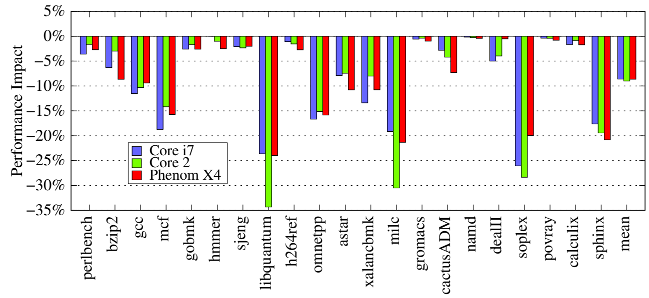

Figure 5: Benchmark slowdown when co-located with `lbm`.

both Google applications and benchmark applications. In addition, applications present various levels of performance sensitivity to such heterogeneity. This indicates that the homogeneity assumption may leave a large performance opportunity untapped.

3.3 OF: An Opportunity Metric

An important concept arises from the previous section. Depending on how "immune" or "sensitive" an application is to microarchitectural and co-runner variation, each application would benefit differently from a job mapping policy that takes advantage of heterogeneity. We introduce a metric, *opportunity factor*, that approximates a given application's potential performance improvement opportunity relative to all other applications, given a particular mix of applications and machine types. The higher the opportunity factor, the more sensitive an application is to diversity in the WSC. Note that this opportunity factor can be calculated only when the application mix and the machine mix are known.

For a given WSC, we can denote the application of type i as A_i, and the microarchitecture of type j as M_j. We define the speedup factor for A_i as:

$$SF_{A_i} = \frac{max_{j,k}\{IPS_{A_i,M_j,C_k}\} - min_{j,k}\{IPS_{A_i,M_j,C_k}\}}{min_{j,k}\{IPS_{A_i,M_j,C_k}\}}, \quad (1)$$

where IPS_{A_i,M_j,C_k} is application A_i's IPS (instruction per second) when it is running on machine M_j with a set of co-runners C_k. The SF_{A_i} is essentially the amount of performance variability of A_i in all possible configurations of the execution environment, composed of the cross product of all machine options and co-runner options. Using SF_{A_i}, we can define the *Opportunity Factor* (OF) for A_i as:

$$OF_{A_i} = \frac{SF_{A_i}}{\sum_j SF_{A_j}} \quad (2)$$

OF_{A_i} represents the sensitivity of each application type to the overall heterogeneity of a given application mix relative to all other applications. This metric allows WSC designers, operators and reliability engineers to reason about the performance improvement potential of various applications in the WSC and identify applications that are most likely to benefit from heterogeneity-aware job mapping. We present and discuss OF results in Section 5.2.

4. WHARE-MAP

In this section, we present an approach to exploit heterogeneity that is particularly well-suited for production WSCs as it leverages already in-place subsystems found in state-of-the-art WSCs to perform job-to-machine mapping.

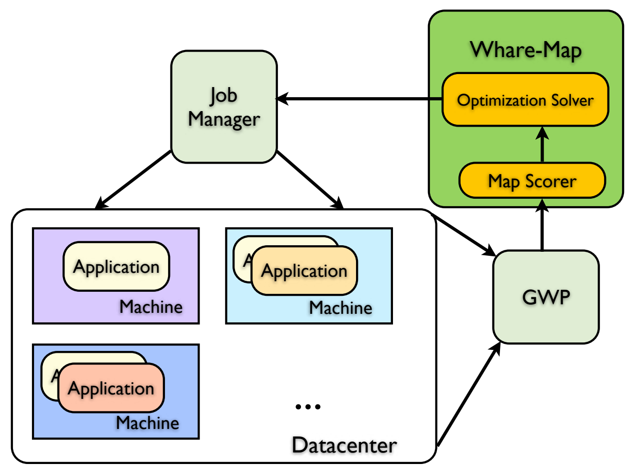

Figure 6: The Overview of Whare-Map

4.1 Overview

Whare-Map harnesses the continuous profiling information provided by in-production monitoring services such as the Google Wide Profiler (GWP) [27] to intelligently map jobs to machines. Figure 6 illustrates how *Whare-Map* is integrated in the core system architecture of WSCs, enabling them to exploit heterogeneity. We formulate the problem of mapping jobs to machines as a combinatorial optimization problem and thus an integral component of Whare-Map is the optimization solver (discussed in Section 4.2).

Another key requirement for *Whare-Map* is the continuous online analysis and comparison of mapping decisions. As illustrated in Figure 6, *Map Scorer* utilizes GWP to perform such analysis. GWP is a system-wide service that continuously monitors and profiles jobs as they run in production WSCs and archives the profiling information in a database. As shown in the figure, *Map Scorer* extracts information from the GWP database to build and continuously refine an internal representation of profiles, analyzing the performance of each application in various execution environments. Using the profiles and scoring by *Map Scorer*, the *Optimization Solver* compares various mapping decisions and their relative performance. It is important to note that during the optimization process, instead of the costly approach of actually mapping jobs to various execution environments to identify the best mapping, the *Optimization Solver* relies on the *Map Scorer* to utilize the historical profiling data from GWP of how well a job performs in each given environment. A continuous profiling service such as

Approach	Description	Complexity
Whare-C	**Co-location Score:** This score is based only on co-location penalty and only requires profiling the co-location penalty on any type of machine. Once a co-location profile is collected it is then used to score that co-location regardless of the underlying microarchitecture.	$\|A\|^n$
Whare-Cs	**Co-location Score (Smart):** This score is based on co-location penalty with microarchitecture specific information. Information about co-location penalty must be collected for all platforms of interest.	$\|A\|^n \times \|M\|$
Whare-M	**Microarchitectural Affinity Score:** This score is based on microarchitectural affinity and captures only the speedup of running each application on one microarchitecture over another.	$\|A\| \times \|M\|$
Whare-MCs	**Microarchitectural Affinity and Co-location Score:** This scoring method includes both microarchitectural affinity and microarchitecture specific co-location penalty. This scoring technique has the heaviest profiling requirements.	$\|A\|^{n+1} \times \|M\|$

Table 5: Scoring Policies for Mapping

GWP is a key component that makes *Whare-Map* feasible in live production systems. It is also important to make the distinction between the costs associated with populating the GWP database (referred to later as profiling complexity) and the cost of utilizing the information in GWP's database to search for the optimal mapping. The former occurs continuously through the lifetime of operation of the WSC, while the latter is often in the order of minutes for a typical scale of thousands of machines and dozens of application types.

4.2 Whare-Map: An Optimization Problem

As mentioned earlier, we formulate the problem of mapping jobs of different types and characteristics to a set of heterogeneous machine resources as a combinatorial optimization problem. The optimization objective is to maximize the overall performance of the entire WSC, i.e., the aggregated instruction-per-second (IPS) of all jobs. This formulation as an optimization problem is especially suitable for modern WSCs for a number of reasons: 1) The set of important applications are known and fairly stable; 2) the main web-service jobs are often long running jobs; and 3) migrations of jobs rarely happens because of the high cost. Our *Whare-Map* is then essentially a solver for this optimization problem.

Algorithm 1: Optimization Algorithm in Whare-Map

```
Input: set of free machines and available jobs
Output: an optimized mapping
while free machines and available jobs do
    map random job to random machine;
end
set last_score to the score of current map;
while optimization timer not exceeded do
    foreach machine do
        foreach job on that machine do
            swap job with random job on random machine;
            set cur_score to the score of current map;
            if mapping score is better then
                set last_score to cur_score;
            else
                swap jobs back to original placements;
            end
        end
    end
end
```

The core algorithm (Algorithm 1) we use to solve the optimization problem is inspired by well established traditional iterative optimization techniques [26, 28, 32]. We use a *stochastic hill climbing* numerical optimization approach in Whare-Map as it is well suited for the type and scale of the problem of mapping jobs to machines and converges rather quickly for our problem formulation (typically less than 1 minute of search). It is important to note that other numerical optimization approaches such as *simulated annealing* and *genetic algorithms* can also be used to perform the search. However, we observe that for the nature of our problem, the stochastic hill climbing approach produces mappings that match the quality of these alternatives and converges quickly.

4.3 Map Scoring

A *score* of a particular placement of a job to a machine is used to measure the quality of the job placement. We use the sum of all individual placement scores to score an entire map of jobs to machines. The higher the score, the better the map. The scoring policy is an essential part in *Whare-Map*. It is used in each optimization iteration to compare mappings. In this work, we present and evaluate a number of scoring policies that vary in the required profiling necessary to generate the score. Table 5 shows the descriptions and profiling complexities for our map scoring policies, where $|A|$ corresponds to the number of application types, $|M|$ corresponds to the number of machine types, and n corresponds to the number of co-runners allowed on each machine. The profiling complexity indicates the amount of profiling the *Map Scorer* needs from GWP. For example, among all policies, Whare-M requires the smallest amount of profiling, $|A|$ x $|M|$, indicating that the scorer only needs performance profiles of each application type on each machine type from GWP without the need of knowing the application's co-runners when the profiling was conducted. In a practical setting of a WSC, $|A|$ is in the order of magnitude of 10s, $|M|$ is often less than 10, and n is often only 1 or 2. Typically, only one or two major web-service jobs, in addition to several low-overhead background processes such as `logsaver`, are co-located on a given machine in a WSC.

The accuracy of the scoring policy determines the mapping quality of *Whare-Map*. Whare-MCs has the complete information and should provide the best result. Meanwhile, Whare-M, Whare-C and Whare-Cs require less time for GWP to collect all needed information. However, they are also less accurate, and thus may lead to suboptimal results. The trade off is between the amount of available profiling information and maximizing the performance gain. In addition, the landscape of diversity present in the WSC has a significant impact on the usefulness of some profiling information. In Section 5.4, we further investigate the factors that impact the diversity and discuss the selection of the appropriate map scoring policies.

The complexity of *Whare-Map* is decided by both the profiling complexity of scoring policies and the computation complexity of the optimization solver. However, as we mentioned before, GWP continuously profiles in the background through the lifetime of a WSC and its cost is thus hidden from *Whare-Map*. *Whare-Map* simply utilizes the profiling information available at any given time and continually refines its performance profiles based on the newly accu-

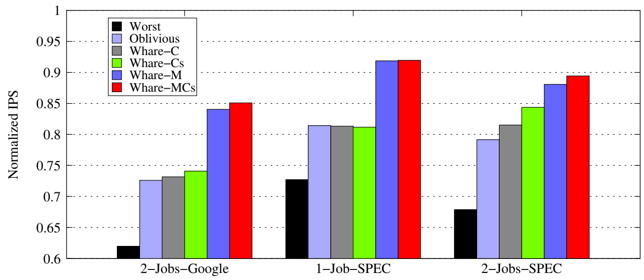

Figure 7: The normalized performance (aggregated IPS) of *Whare-Map*, compared to the heterogeneity-oblivious mapper and the worst case (higher is better)

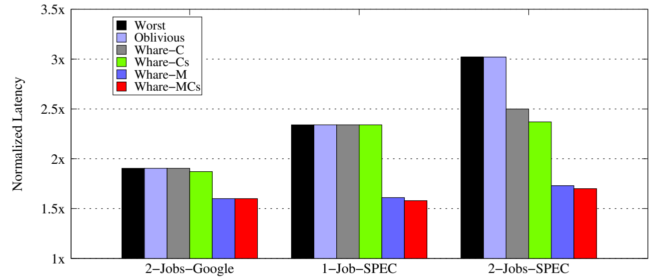

Figure 8: The normalized latency of a given WSC when using *Whare-Map*, compared to the heterogeneity-oblivious mapper and worst case (lower is better)

mulated profiling data collected by GWP to continuously improve its scoring and thus the mapping decisions. On the other hand, the complexity of using our optimization solver based on the map scores to search for the optimal mapping is relatively low, typically in an order of seconds.

5. EVALUATING WHARE-MAP

In this section, we measure the performance improvement when using our heterogeneity-aware approach, *Whare-Map*, over current heterogeneity-oblivious mapping. We also evaluate the performance of four different scoring policies discussed in Section 4.3. In addition to the overall performance of an entire WSC, we present the application-level performance achieved by *Whare-Map* and compare it with the estimation provided by the opportunity factor. Lastly, we delve into the factors affecting emergent heterogeneity and how this heterogeneity impacts the cost efficiency of server purchase decisions.

5.1 Experimental Methodology

We conduct thorough investigation and evaluation in three domains. We evaluate Whare-Map using Google and benchmark testbeds (Section 5.2). In addition, we analyze its potential in production WSCs running live web-services (Section 5.3). For experimentation using Google and benchmark testbeds, we use platform types previously presented in Tables 3 and 4 along with the 9 Google key applications and 22 SPEC CPU2006 benchmarks, respectively.

For our testbed evaluation, we construct an oracle based on comprehensive runs on real machines. Given a map of jobs to a set of machines, this oracle reports the performance of that mapping. To construct this oracle, we run all combinations of co-locations on all machine platforms and collect performance information. This performance information is in the form of instructions per second (IPS) for each application in every execution environment. Using this information we construct a *knowledge bank* that is used as a reference for the performance of a particular event in a given WSC. We use the same Google workload suite used to evaluate machine configurations internally. These workloads are composed of Google's core commercial services and have been tuned to exercise mainly the processor and the memory subsystems and have minimal run to run performance variance (˜1% on average). The input set used is composed of large traces of real world production queries. This setup allows us to focus our study on the emergent heterogeneity in microarchitectural configuration and the microarchitectural interaction between co-runners. The knowledge bank serves two purposes for the evaluation conducted on the testbeds. Firstly, given a job-to-machine mapping, we use the knowledge bank as the oracle to calculate the aggregate performance of the entire WSC composed of various machine types. Secondly, the knowledge bank is used to model GWP where, depending on the scoring policy, partial information (such as only machine heterogeneity or co-location heterogeneity) is used for various levels of profiling complexity. In our production analysis, live GWP information is used.

5.2 Google and Benchmark Testbeds

[Overall IPS] In Figure 7, we compare our *Whare-Map*, the heterogeneity-oblivious mapper and the worse case mapper for overall performance of a WSC. The heterogeneity-oblivious mapper randomly maps a job to a machine based only on the job's resource requirement irrespective to the heterogeneous machine types and corunning jobs on that machine. We first use the aggregated instructions per second (IPS) of all machines as our performance metric for the entire WSC. The experiments shown in this figure are conducted on Google testbed (1st cluster of bars) and benchmark testbed (2nd and 3rd clusters of bars). The y-axis shows the normalized overall performance (IPS) of a WSC when using various job-to-machine mapping policies. To calculate the normalized IPS performance of an entire WSC for a given job-to-machine mapping, we aggregate the average IPS of all jobs. The normalization baseline for each cluster of bars is the sum of the average IPS of each job when it is run alone on its best performing machine type. Higher is better.

The first cluster of bars presents results for the Google testbed. In this experiment, the testbed WSC is composed of 500 machines with 1000 jobs running; two jobs are co-located on each machine. We choose the 2-Jobs scenario because, given typical core and memory requirements, one or two major web-service jobs are co-located on a given machine in our production WSCs. The machine composition and workloads of the WSC are randomly selected from the three machine types shown in Table 3 and 9 key Google applications shown in Table 2. Each bar in the cluster presents the performance for the worst mapping, heterogeneity-oblivious mapping, as well as *Whare-Map* using four varying scoring policies as discussed in Section 4.3. Similarly, the second and third clusters present results for benchmark testbed.

For the 1-Job scenario, there are 500 jobs running in a WSC composed of 500 machines with one job running on each machine; while the 2-Jobs scenario has 1000 jobs running on 500 machines. The machine composition and workloads of the WSC are also randomly generated using the three machine types from our benchmark testbed (Table 4) and SPEC CPU2006 suite.

In Figure 7, we observe a significant benefit from using

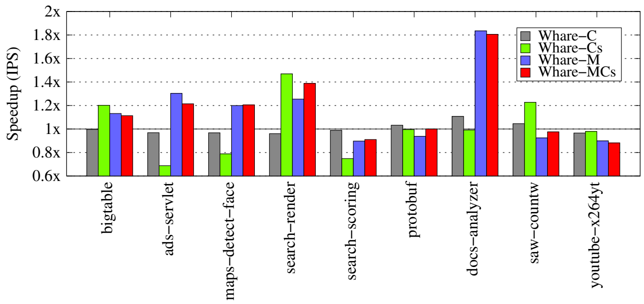

Figure 9: Speedup at the application level over heterogeneity-oblivious.

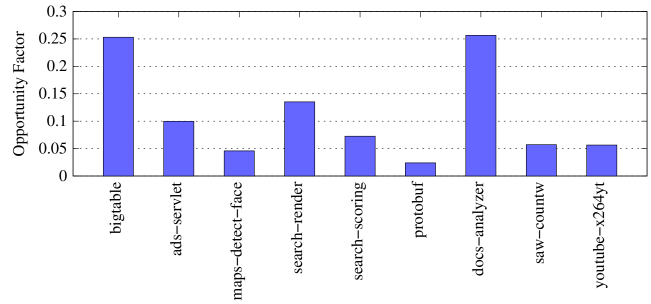

Figure 10: Opportunity factor of each application.

Whare-Map for the Google testbed experiment. Among four scoring policies of *Whare-Map*, we achieve the best performance when considering both machine and co-location heterogeneity (Whare-MCs), which improves the overall normalized IPS of the entire WSC by 18% over the heterogeneity-oblivious mapping (from 0.72x to 0.85x) and 37% over the worst case mapping. Also, in this experiment, Whare-M performs comparably well as Whare-MCs. This indicates that there is a significant performance benefit when considering the machine heterogeneity. Meanwhile, when only co-location effects are considered to score maps (Whare-C and Whare-Cs) we observe less overall performance gains. It is within 1-2% of the heterogeneity-oblivious mapping result as Whare-C and Whare-Cs focus only on the performance impact of resource contention between co-located applications on the same machine. Note that the heterogeneity-oblivious mapping already greatly improves the IPS over the worst case, by around 17%. When the workload is a fairly balanced mix of contentious (memory-intensive) applications and non-contentious (CPU intensive) applications, randomizing the mapping can effectively decrease the chance of co-locating two contentious applications and in turn improve over the worst case by reducing a significant amount of co-location penalties. These results indicate that for our Google workload and production machine mix in our testbed, exploiting the machine heterogeneity may have a bigger impact than considering co-location heterogeneity alone. However the relative importance of machine and co-location heterogeneity depends on the machine/workload mix. We explore those impacting factors in greater detail in Section 5.4.1.

The results for benchmark testbed, shown as the second and third clusters of bars, are in general consistent with the Google testbed results. For the 1-Job scenario in the benchmark testbed, as we expect, Whare-C and Whare-Cs do not improve performance over heterogeneity-oblivious mapping while Whare-M and Whare-MCs perform equally well. This is because there is no co-location in a 1-Job scenario. The performance improvement of *Whare-Map* using Whare-MCs over the worst case mapping is 26% and close to 14% over heterogeneity-oblivious mapping. For the 2-Jobs scenario (the 3rd cluster) we observe that scoring policies that only consider co-location heterogeneity (Whare-C, Whare-Cs) are quite effective, generating up to an 8% improvement over heterogeneity-oblivious mapping. This is better than the performance of Whare-C in 2-Jobs scenarios for Google applications, demonstrating that the effectiveness of Whare-C depends on the machine/workload mix. Only considering microarchitectural heterogeneity without considering co-location (Whare-M) can produce 12% performance benefit over the heterogeneity-oblivious mapping, higher than Whare-C. When *Whare-Map* combines both machine heterogeneity and co-location penalty heterogeneity (Whare-MCs), the performance improvement is increased to about 16%.

[Latency] In addition to the aggregated IPS, we also compare the latency of all jobs in a WSC, defined as the execution time of the longest-running job under a given job-to-machine mapping. Figure 8 shows the latency of various mapping policies, normalized to the latency when all jobs run alone on their best performing machine type. Interestingly, although the heterogeneity-oblivious mapping can improve the average IPS performance, it performs equally poorly as the worst mapping for improving latency. In this experiment our *Whare-Map* improves the job placement of the slowest job resulting in lower overall latency. Again, Whare-MCs performs the best, and Whare-M performs comparably well.

[Opportunity Factor] We further examine the performance improvement of each application and compare it with the estimation by the *opportunistic factor* (Section 3.3). Figure 9 presents the performance improvement at the application level from the 2-Jobs scenario with 500 machines and 1000 applications for the Google testbed as in Figures 7 and 8. The y-axis shows each application type's performance using *Whare-Map*, normalized to each type's average performance in a heterogeneity-oblivious mapping. This figure demonstrates that application types have varying amounts of performance benefit from *Whare-Map*. For example, while there is a 16%-19% performance improvement overall, `docs-analyzer`, which is sensitive to both microarchitectural and co-location heterogeneity, achieves a 80% performance improvement over heterogeneity-oblivious mapping.

There are also applications that suffer performance degradation. However, as shown in the figure, the performance improvement greatly outweighs these degradations. In the cases where machine heterogeneity is considered (Whare-M, and Whare-MCs), these degradations are negligible. Figure 10 presents the opportunity factor (OF) of each application, calculated using Equation 1 and Equation 2 in Section 3.3. As the corresponding Figures 9 and 10 show, OF is an effective metric in correctly identifying how sensitive various applications are to heterogeneity. However, not all of the application-level opportunity is realized. Remember that mapping to exploit WSC heterogeneity is a constraint optimization problem. As a result, not all applications can be mapped to their individual optimal situations to achieve the maximum performance improvement. For example, `docs-analyzer` has a slightly better OF than `bigtable` and they both prefer the Westmere platform, so as the "preferred" Westmeres in a WSC are consumed by `docs-analyzer`, `bigtable`'s mapping options are reduced.

Keep in mind that Whare-Map and OF serve different

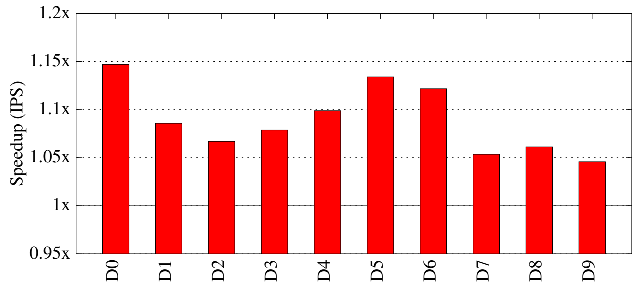

Figure 11: Performance improvement from *Whare-Map* over the currently deployed mapper in production

purposes. While Whare-Map exploits heterogeneity, OF is a predictor as to how applications will be affected by heterogeneity. OF is a much needed metric for WSC operators for understanding a key property of the applications within the eco-system of the WSC.

5.3 Whare-Map in the Wild

Lastly, we employ *Whare-Map* on production performance profiles to study the potential performance improvement when exploiting heterogeneity in the wild for live production WSCs. We conducted our evaluation in the same 10 randomly selected WSCs shown in Table 1. These WSCs present various levels of machine heterogeneity. We have collected detailed Google-Wide Profiling (GWP) [27] profiles of around 100 job types running across these WSCs consisting of numerous machines in the wild. These jobs span most of Google's main products, including websearch. Using the GWP profiles, we conducted a postmortem *Whare-Map* analysis to re-map jobs to machines and calculate the expected performance improvement. Firstly, instructions per cycle (IPC) samples are derived from GWP profiles. We use *cycle* and *instruction* samples collected using hardware performance counters by GWP over a fixed period of time, aggregated per job and per machine type. IPS (instructions per second) for each application on each machine type is then computed by normalizing the IPC by the clock rate. These IPS samples are used for map scoring. Here we use Whare-M policy, considering only microarchitectural heterogeneity.

Using our *Whare-Map* approach, we produce an intelligent job-to-machine mapping in a matter of seconds at the scale of thousands of machines of multiple types, over a hundred job types and the performance profiles of over the course of a month of operation. Figure 11 shows the calculated performance improvement when using *Whare-Map* over the currently deployed mapping in 10 of Google's active WSCs. Even though some major applications are already mapped to their best platforms through manual assignment, we have measured significant potential improvement of up to 15% when intelligently placing the remaining jobs. This performance opportunity calculation based on this paper is now an integral part of Google's WSC monitoring infrastructure. Each day the number of 'wasted cycles' due to inefficiently mapping jobs to the WSC is calculated and reported across each of Google's WSCs world wide.

5.4 Factors Impacting Heterogeneity in WSCs

The rationale behind the homogeneity assumption stems from a lack of understanding on how the gradual introduction of diversity in a WSC impacts performance variability. In this section, we perform a study of how varying the diversity in a WSC affects the performance opportunity from the heterogeneity available in the WSC along two dimensions, application mix and machine platform mix. We then present insights into how these two factors affect server purchase options as well as the selection of the appropriate map scoring policies.

5.4.1 Impact of Workload Mix on Heterogeneity

In this section, we evaluate a variety of workload mixes to investigate how workload mixes impact the performance improvement when exploiting the heterogeneity. We partitioned our 9 Google applications into two types, memory intensive (and thus likely to be contentious) and CPU intensive. We also selected the top 8 memory intensive applications and the top 8 CPU intensive applications from SPEC 2006. As shown in Table 6, we constructed 7 types of workloads using our classification. We then conducted various job mapping experiments on these workloads to investigate the performance benefit of using *Whare-Map* over the heterogeneity-oblivious mapping. All experiments on the Google testbed use a WSC of 500 machines evenly distributed from 3 machine types listed in Table 3 (166 Clovertown Xeon, 166 Istanbul Opteron, 168 Westmere Xeon). Similarly, the benchmark testbed experiments use 400 machines composed of 3 types of microarchitectures listed in Table 4 (133 Core i7s, 133 Core 2s, 134 Phenom X4s).

Figures 12 and 13 present our experimental results for the Google and benchmark testbed, respectively. We conducted a 2-jobs-per-machine experiment using Google testbed and both 1-Job and 2-Jobs scenarios for the benchmark testbed. In each figure, the x-axis shows each experiment's configurations. For example, in Figure 13, the notation 1J-MostlyCPU indicates the 1-job-per-machine scenario and the workload is composed of $\frac{3}{4}$ CPU intensive benchmarks and $\frac{1}{4}$ memory intensive benchmarks. The y-axis shows Whare-Map's performance improvement compared to the heterogeneity-oblivious mapping. The performance metric is the overall aggregated IPS of all machines. As the figures show, the amount of performance benefit of using *Whare-Map* to take advantage of heterogeneity varies when the workload mix varies. Specifically, we have the following observations and insights.

1) The performance benefit potential is smaller for CPU intensive workloads than memory intensive workloads or mixed workloads. Figure 12 shows that for Google experiments, the workload of mostly CPU intensive applications achieves just over a 10% improvement over the heterogeneity-oblivious mapping, as opposed to close to 15% for memory intensive workloads. In Figure 13, both 1J-CPU and 2J-CPU experiments have relatively low performance improvement (less than 5%). This indicates that for CPU intensive benchmarks, the microarchitectural heterogeneity is smaller. For our workloads and the 2 sets of microarchitectures (Tables 3 and 4), **much of the performance variability and opportunity are in the memory subsystem heterogeneity**.

2) In general, more diverse workloads, such as workloads composed of both CPU and memory intensive benchmarks, have higher performance improvement potential for using *Whare-Map* than workloads composed of pure CPU or pure memory intensive benchmarks. For example, in Figure 13, for the 1-Job scenarios (left half of the figure), Whare-MCs has more performance improvement over the heterogeneity-oblivious mapping for 1J-mix (15%) than 1J-CPU (3%) or 1J-Memory (10%). Similarly, for the 2-Jobs scenarios (right half of Figure 13), when the workload is composed of only CPU intensive benchmarks (2J-CPU), the performance improvement is much smaller (4%) than that for 2J-mix (14%), which has a more heterogeneous workload.

3) Considering machine heterogeneity only (Whare-M) is fairly competitive with considering both machine and co-

Workload	Application Types
Google Mostly Mem	bigtable, ads-servlet, search-render, docs-analyzer
Google Mostly CPU	maps-detect-face, search-scoring, protobuf, saw-countw, youtube-x264yt
Memory	lbm, libquantum, mcf, milc, omnetpp, soplex, sphinx, xalancbmk
CPU	hmmer, namd, povray, h264ref, gobmk, dealII, sjeng, perlbench
Mix ($\frac{1}{2}$Mem/ $\frac{1}{2}$CPU)	lbm, libquantum, mcf, milc, hmmer, namd, povray, h264ref
Mostly Mem ($\frac{3}{4}$Mem/$\frac{1}{4}$CPU)	lbm, libquantum, mcf, milc, omnetpp, soplex, hmmer, namd
Mostly CPU ($\frac{3}{4}$CPU/$\frac{1}{4}$Mem)	hmmer, namd, povray, h264ref, gobmk, dealII, lbm, libquantum

Table 6: Workload Mixes

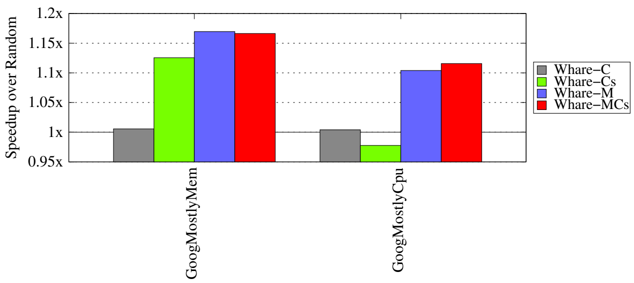

Figure 12: Impact of varying workload mix on available heterogeneity for the Google testbed. Performance is normalized to heterogeneity-oblivious mapping (higher is better)

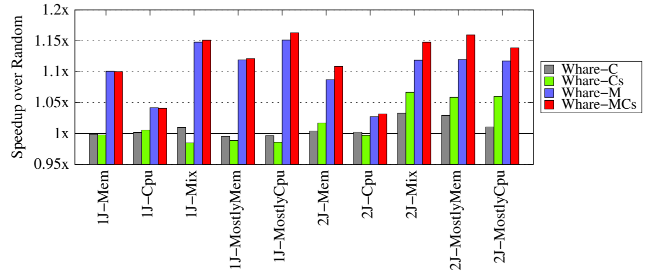

Figure 13: Impact of varying workload mix on available heterogeneity for SPEC benchmark testbed. Performance is normalized to heterogeneity-oblivious mapping.

location heterogeneity (Whare-MCs) in most scenarios. On the other hand, considering co-location only (Whare-Cs) does not outperform considering machine heterogeneity only (Whare-M) in any scenario. One reason is that for both our Google and benchmark testbeds, the performance variability due to microarchitectural heterogeneity is as high as 3.5x and 2x, respectively, while the performance variability due to the penalty of co-locating two jobs is only around 30% (Figures 2 and 3). However, in the next section we will further investigate the performance difference between Whare-MCs and Whare-M when the amount of microarchitectural heterogeneity changes.

5.4.2 *Impact of Machine Mix on Heterogeneity*

In addition to the workload mix, microarchicture mix also has a significant impact on the amount of the heterogeneity in a WSC. In this section we study the impact of varying microarchitecture mix on the performance improvement of *Whare-Map*. We conducted experiments using 6 types of machine mixes for the Google testbeds. The 6 types include: an entire WSC composed of all Clovertown Xeon, all Istanbul Opteron, all Westmere Xeon, $\frac{1}{2}$ Clovertown + $\frac{1}{2}$ Istanbul, $\frac{1}{2}$ Instanbul + $\frac{1}{2}$ Westmere and $\frac{1}{2}$ Clovertown + $\frac{1}{2}$ Westmere. The workloads used for the Google testbed is composed of all 9 key Google applications (Table 2). We also conducted similar experiments on the benchmark testbed, using a a workload composed of mostly memory intensive applications (Table 6).

Figures 14 and 15 present the results for the Google and benchmark testbed, respectively. Similar to previous figures in Section 5.4.1, in each figure, the y-axis shows the performance improvement of *Whare-Map* using four different scoring policies over the heterogeneity-oblivious mapping for different machine mixes.

The first observation from these two figures is that even mixes of machines from a similar generation present a significant performance opportunity for exploiting heterogeneity. In Figure 14, even for machine mixes composed of only 2 types of machines, *Whare-Map* generates significant performance improvement over heterogeneity-oblivious mapping. For Clovertown and Istanbul, which have similar average performance (Figure 2), the performance improvement of their mix is also significant (more than 10%). Similar observations can be made for the benchmark testbed as shown in Figure 15.

It is also important to note that for some machine mixes, the benefit of using Whare-MCs over Whare-M is significant. For example, in the 2J-Core 2+Phenom X4 scenario shown in Figure 15 (the last cluster of bars), Whare-MCs's performance improvement over the heterogeneity-oblivious mapping is 14%, significantly higher than the Whare-M's 8% improvement. This is different from the observations we made in Section 5.4.1 that often Whare-M performs similarly with Whare-MCs. The reason for this difference is that there is less microarchitectural heterogeneity (only 2 types of machines in the mix) in these experiments than those in Section 5.4.1 and thus the co-location heterogeneity becomes more important. This observation demonstrates that although the microarchitectural heterogeneity is generally dominantly important, the amount of additional performance benefit when considering co-location is largely determined by the workloads mix and the machine mix. We discuss more on this topic in Section 5.6.

5.5 Which Servers to Purchase?

Important questions arises when making server purchasing decisions: Is heterogeneity in a WSC desirable or not? Should it be increased or decreased?

The heterogeneity study in this paper indicates that when making such heterogeneous vs. homogeneous decisions, simply comparing servers' average performance for a workload suite is insufficient and may be misleading. Instead, we advocate using *Whare-Map* to estimate the performance of WSCs with various machine mixes. In fact, Whare-Map makes the heterogeneous WSC a potentially more cost ef-

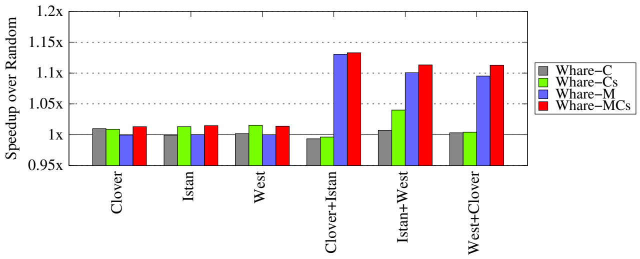

Figure 14: Impact of varying machine mix on heterogeneity for the Google testbed. Performance is normalized to heterogeneity-oblivious mapping(higher is better).

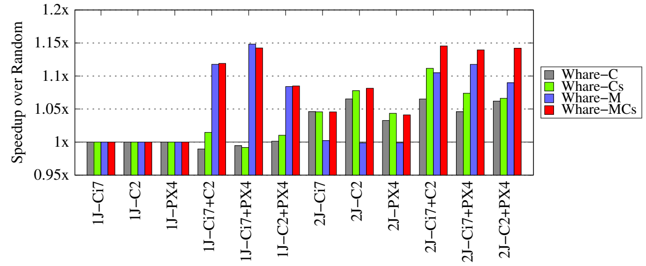

Figure 15: Impact of varying machine mix on heterogeneity for SPEC benchmark testbed. Performance is normalized to heterogeneity-oblivious mapping.

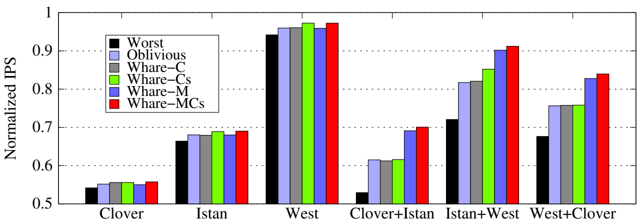

Figure 16: Normalized performance of various options of WSC machine composition(higher is better).

fective (better performance/dollar) option than purely homogeneous WSCs.

To illustrate that, Figure 16 shows the performance of several WSCs composed of various machine mixes. The experiments are conducted using the same workload as in Figure 14, composed of all 9 Google applications. In contrast to Figure 14, the performance of all experiments here is normalized to a single baseline, the aggregate IPS of all applications, each running alone on its best performing platform. The baseline thus is an upper bound on performance. This facilitates the comparison of relative performance between WSCs. The key observation in this graph is highlighted when comparing the all `Istan` cluster with the half `Istan` and half `Clover` clusters. The `Istan` cluster is more expensive as it is composed of the newer generation. However, when using Whare-Map to place jobs where they run best, the cheaper cluster, *Istan+Clover*, performs just as well (and indeed a bit better as some jobs actually prefer the Clover machine).

5.6 Revisiting Map Scoring

In addition to the findings discussed in the above sections, this study also leads to a number of insights on how to select the scoring policy:

1) **No free lunch**. Among all four scoring policies, Whare-MCs always delivers the best performance improvement. However, it also requires the most amount of profiling to be effective.

2) **Whare-M: big bang for your buck**. As this section shows, in most settings, Whare-M generates significant performance improvement over heterogeneity-oblivious mapping with a very small amount of profiling. The profiling complexity is only $|A|$x$|M|$ as shown in Table 5. This indicates that Whare-M can be adopted as an easy and effective first step for *Whare-Map* and can be triggered as soon as GWP finishes profiling the basic machine heterogeneity information.

3) **Whare-MCs: gradually improve over Whare-M.** As Section 5.4.2 shows, depending on the workload and machine mixes, Whare-MCs may also improve over Whare-M significantly, delivering extra performance benefit, especially when there is much co-location penalty variability. Therefore, Whare-MCs can be used to gradually improve over the mapping of Whare-M, as the GWP accumulates more information regarding co-location.

4) **Continuous Knowledge Refinement** It is important to remember that although the profiling complexity of Whare-MCs appears high (Table 5), GWP runs continuously throughout the lifetime of the WSC probing each machine once every minute. As the scale of the WSC increases to thousands of machines the rate at which the profiling information becomes robust also increases.

6. RELATED WORK

Perhaps the most closely related works are those focused on heterogeneity in datacenters that have appeared in both the systems and architecture communities [2, 9, 12, 13, 36]. Our work is complimentary to these works in that the assumptions that underlie these works apply to systems providing utility computing and/or does not leverage in production continuous profiling subsystems such as GWP. In contrast to our work, datacenters providing utility computing can not make the assumption that applications and services running in these datacenters are known a priori. Also, in contrast to the prior work on MapReduce, our work extends the core architecture of WSCs at the abstraction layer closest to the underlying hardware, is general across various programming paradigms, and leverages continuous profiling subsystems as opposed to performing trials.

There is much related research on datacenters focused on improving energy efficiency [1, 3, 6, 16, 19, 20, 22, 24, 25, 34]. There has also been work on scheduling in datacenters [17], enabling QoS-aware control in datacenters [23, 29, 31], and programming datacenters [10]. A recent work that shares some similarities with our work presents PROPHET, a goal-oriented provisioning infrastructure that tunes the WSC to satisfy the needs of particular end users [33]. The research work that is closest to ours discusses a scheduling policy which uses a linear programming problem that maximizes system capacity to map an application across a desktop grid [4]. This work focuses on distributed desktop computers and does not consider the interaction between microarchitectural and co-location heterogeneity. There has been a significant amount work in domain of heterogeneous multicore that is also related to this work such as the work by

Winter et al. [32] that investigated the task of scheduling for "unpredictably" heterogeneous multicore processors due to process variation.

7. CONCLUSION

In this work, we examine the WSC as a heterogeneous system and show that emergent heterogeneity must be considered when mapping jobs to machines. We investigate microarchitectural heterogeneity in the WSC and find that even when considering platforms from competing generations, there is significant and idiosyncratic variability across applications; and, application co-location is particularly important when considering the heterogeneous WSC. In this work, we also demonstrate how WSC heterogeneity can be exploited and investigate how varying the application mix as well as the machine mix can impact performance. We find that for applications that are sensitive to variations in microarchitecture and co-runners, we observe a performance improvement of up to 80% when employing our approach over current random scheduling techniques. Even in a WSC composed entirely of state-of-the art machines, we can improve the overall performance by 18%. We also present a case study from a live WSC confirming this result, demonstrating up to 15% performance improvement.

8. REFERENCES

[1] D. Abts, M. Marty, P. Wells, P. Klausler, and H. Liu. Energy proportional datacenter networks. *ISCA '10*, Jun 2010.

[2] F. Ahmad, S. T. Chakradhar, A. Raghunathan, and T. N. Vijaykumar. Tarazu: optimizing mapreduce on heterogeneous clusters. ASPLOS '12, pages 61–74, New York, NY, USA, 2012. ACM.

[3] F. Ahmad and T. Vijaykumar. Joint optimization of idle and cooling power in data centers while maintaining response time. *ASPLOS '10*, Mar 2010.

[4] I. Al-Azzoni and D. Down. Dynamic scheduling for heterogeneous desktop grids. *GRID '08*, Sep 2008.

[5] G. Banga, P. Druschel, and J. C. Mogul. Resource containers: a new facility for resource management in server systems. In *OSDI '99*, Berkeley, CA, USA, 1999. USENIX Association.

[6] L. A. Barroso, J. Dean, and U. Hölzle. Web search for a planet: The google cluster architecture. *IEEE Micro*, 23(2):22 – 28, 2003.

[7] L. A. Barroso and U. Hölzle. The datacenter as a computer: An introduction to the design of warehouse-scale machines. *Synthesis Lectures on Computer Architecture*, pages 1–120, Sep 2009.

[8] L. A. Barroso and P. Ranganathan. Guest editors' introduction: Datacenter-scale computing. *IEEE Micro*, 30:6–7, 2010.

[9] J. Burge, P. Ranganathan, and J. Wiener. Cost-aware scheduling for heterogeneous enterprise machines (cash'em). *CLUSTER '07*, Sep 2007.

[10] S. Bykov, A. Geller, G. Kliot, J. Larus, R. Pandya, and J. Thelin. Orleans: A framework for cloud computing. Technical Report MSR-TR-2010-159, Microsoft Research, November 2010.

[11] F. Chang, J. Dean, S. Ghemawat, W. Hsieh, D. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. Gruber. Bigtable: a distributed storage system for structured data. *OSDI '06*, Nov 2006.

[12] B.-G. Chun, G. Iannaccone, G. Iannaccone, R. Katz, G. Lee, and L. Niccolini. An energy case for hybrid datacenters. *SIGOPS Oper. Syst. Rev.*, 44(1):76–80, Mar. 2010.

[13] C. Delimitrou and C. Kozyrakis. Paragon: QoS-Aware Scheduling for Heterogeneous Datacenters. In *Proceedings of the Eighteenth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, March 2013.

[14] EPA. Epa report to congress on server and data center energy efficiency. Technical report, U.S. Protection Agency, 2007.

[15] J. Hamilton. Internet-scale service infrastructure efficiency. *SIGARCH Comput. Archit. News*, 37(3):232–232, 2009.

[16] T. Heath, B. Diniz, E. V. Carrera, W. Meira, Jr., and R. Bianchini. Energy conservation in heterogeneous server clusters. In *Proceedings of the tenth ACM SIGPLAN symposium on Principles and practice of parallel programming*, PPoPP '05, pages 186–195, New York, NY, USA, 2005. ACM.

[17] R. Huang, H. Casanova, and A. Chien. Automatic resource specification generation for resource selection. *SC '07*, Nov 2007.

[18] K. Lim, P. Ranganathan, J. Chang, C. Patel, T. Mudge, and S. Reinhardt. Understanding and designing new server architectures for emerging warehouse-computing environments. In *Proceedings of the 35th Annual International Symposium on Computer Architecture*, ISCA '08, pages 315–326, Washington, DC, USA, 2008. IEEE Computer Society.

[19] J. Mars, L. Tang, and R. Hundt. Heterogeneity in "homogeneous" warehouse-scale computers: A performance opportunity. *IEEE Computer Architecture Letters*, 2011.

[20] J. Mars, L. Tang, R. Hundt, K. Skadron, and M. L. Soffa. Bubble-up: Increasing utilization in modern warehouse scale computers via sensible co-locations. In *MICRO '11: Proceedings of The 44th Annual IEEE/ACM International Symposium on Microarchitecture*, New York, NY, USA, 2011. ACM.

[21] A. K. Mishra, J. L. Hellerstein, W. Cirne, and C. R. Das. Towards characterizing cloud backend workloads: insights from google compute clusters. *SIGMETRICS Perform. Eval. Rev.*, 37:34–41, March 2010.

[22] R. Nathuji, C. Isci, and E. Gorbatov. Exploiting platform heterogeneity for power efficient data centers. *ICAC '07: Proceedings of the Fourth International Conference on Autonomic Computing*, Jun 2007.

[23] R. Nathuji, A. Kansal, and A. Ghaffarkhah. Q-clouds: managing performance interference effects for qos-aware clouds. *EuroSys '10*, Apr 2010.

[24] S. Pelley, D. Meisner, P. Zandevakili, T. Wenisch, and J. Underwood. Power routing: dynamic power provisioning in the data center. *ASPLOS '10*, Mar 2010.

[25] V. Reddi, B. Lee, T. Chilimbi, and K. Vaid. Web search using mobile cores: quantifying and mitigating the price of efficiency. *ISCA '10*, Jun 2010.

[26] C. R. Reeves, editor. *Modern heuristic techniques for combinatorial problems*. John Wiley & Sons, Inc., New York, NY, USA, 1993.

[27] G. Ren, T. Moseley, E. Tune, S. Rus, and R. Hundt. Google-wide profiling: A continuous profiling infrastructure for datacenters. *IEEE Micro*, 2010.

[28] S. M. Sait and H. Youssef. *Iterative Computer Algorithms with Applications in Engineering: Solving Combinatorial Optimization Problems*. IEEE Computer Society Press, Los Alamitos, CA, USA, 1999.

[29] L. Tang, J. Mars, and M. L. Soffa. Compiling for niceness: Mitigating contention for qos in warehouse scale computers. In *CGO '12: Proceedings of the 2012 International Symposium on Code Generation and Optimization*, New York, NY, USA, 2012. ACM.

[30] L. Tang, J. Mars, N. Vachharajani, R. Hundt, and M. L. Soffa. The impact of memory subsystem resource sharing on datacenter applications. In *Proceeding of the 38th annual international symposium on Computer architecture*, ISCA '11, pages 283–294, New York, NY, USA, 2011. ACM.

[31] L. Tang, J. Mars, W. Wang, T. Dey, and M. L. Soffa. Reqos: Reactive static/dynamic compilation for qos in warehouse scale computers. In *ASPLOS '13: Proceedings of the 18th International Conference on Architectural Support for Programming Languages and Operating Systems*. ACM, 2013.

[32] J. A. Winter and D. H. Albonesi. Scheduling algorithms for unpredictably heterogeneous cmp architectures. *DSN 2008*, pages 42 – 51, 2008.

[33] D. Woo and H.-H. Lee. Prophet: goal-oriented provisioning for highly tunable multicore processors in cloud computing. *SIGOPS Operating Systems Review*, 43(2), Apr 2009.

[34] H. Yang, A. Breslow, J. Mars, and L. Tang. Bubble-pipo: Precise on- line qos management for increased utilization in warehouse scale computers. In *ISCA '13: Proceedings of the 40th annual International Symposium on Computer Architecture*. IEEE/ACM, 2013.

[35] S. Yeo and H.-H. Lee. Using mathematical modeling in provisioning a heterogeneous cloud computing environment. *Computer*, 44(8):55 –62, aug. 2011.

[36] M. Zaharia, A. Konwinski, A. Joseph, R. Katz, and I. Stoica. Improving mapreduce performance in heterogeneous environments. *OSDI'08*, Dec 2008.